REMARKS

In this Office Action, the Examiner objected to the SPECIFICATION because it contains an embedded hyperlink and/or other form of browser-executable code and rejected Claims 1 – 23 under 35 U.S.C. §103(a) as being unpatentable over Silver et al. in view of York et al.

Applicants have amended the SPECIFICATION to delete all instances of an embedded hyperlink and/or other form of browser-executable code therein. Hence, Applicants kindly request withdrawal of the objection to the SPECIFICATION.

Independent Claims 1, 8, 16 and 20 as well as dependent Claims 2 – 4, 6, 7, 10, 11, 15 and 18 were amended to better claim the invention. By this amendment, Claims 1 – 23 remain pending in the Application. For the reasons stated more fully below, Applicants submit that the pending claims are allowable over the applied references. Hence, reconsideration, allowance and passage to issue are respectfully requested.

As disclosed in the SPECIFICATION, an input method editor (IME) is a software program that interprets user operations, such as keystrokes, speaking, or writing using a pen device, to generate text input. For example, an IME may allow a user to type a sequence of keystrokes on a regular English (United States) keyboard to form complex characters found in languages such as Chinese, Japanese, or Korean. The IME uses a character encoding format to associate the user's keystrokes with a "code point", and the code point with a formed character or "glyph" (i.e. the actual writing mark normally associated with the code point).

Over the years, many character encoding formats have been developed. One example is the American Standard Code for Information Interchange, commonly known as "ASCII." ASCII allows characters to be represented by numbers. For example, if the encoding format is decimal, the glyph 'A' in ASCII corresponds to '65'. Other character encoding formats include "EBCDIC"

(Extended Binary-Coded Decimal Interchange Code) developed by IBM, "CCITT" developed by the International Telegraph and Telephone Consultative Committee (now known as the International Telecommunication Union), and "ISO 8859-1" developed by the International Organization for Standardization.

More recently, the "Unicode" encoding format, developed by the Unicode Consortium, has been gaining wider acceptance. Unicode is touted as an "international" character encoding format with enough resolution to notionally provide a unique numeric value or code point for every character, independent of the platform, program, and language. Unicode has been widely adopted in Internet browsers, and is supported in modern platform independent programming languages, such as Java.

In the Unicode encoding system, each character receives a unique Unicode code point having a value in the hexadecimal range 000000 to 10FFFF. Thus, each Unicode code point may be expressed using 21 bits. (There are, however, several Unicode encoding format standards in use: UTF-8, UTF-16, and UTF-32. UTF-8 represents Unicode code points in "code units" of 8 bits. UTF-16 represents Unicode code points in code units of 16 bits. UTF-32 represents Unicode code points in code units of 32 bits. In UTF-32, each Unicode code point is stored in a single code unit.)

Presently, IME developers have had to create IMEs that are platform independent but operating system specific. That is, although the IMEs can run on any hardware platform, they nonetheless depend on the operating system running on the platforms. Hence a need exists for a more efficient solution. The present invention provides such an efficient solution.

According to the teachings of the invention, one or more keystrokes may be used to form a character sequence. This character sequence, which may be entered at a US keyboard, may represent one character, a desired character in Japanese, for example (see paragraphs [0044], [0047], [0058]). Tthe sequence of characters will then be converted to a particular Unicode code point (see

paragraph [0049]). This Unicode code point is the one that corresponds to the desired character (see paragraph [0051]).

The invention is set forth in claims of varying scopes of which Claim 1 is illustrative.

> 1. A method of accessing a platform independent input method editor (IME) from an underlying operating system, comprising:
> receiving keystrokes at an operating system-based input;
> forming a character sequence from said received keystrokes, the character sequence being rendered as a desired character;
> ***passing the character sequence to an IME device driver, the IME device driver converting the character sequence into a first encoding format***;
> ***passing the character sequence encoded in the first format to an operating system-based IME service module***;
> ***from the operating system-based IME service module, calling said platform independent IME to convert said character sequence to a corresponding code point in a second encoding format, the code point being a value assigned to the desired character in a Specification of the second encoding format***; and
> transferring said code point to an operating system-based output. (Emphasis added.)

The Examiner rejected the claims under 35 U.S.C. §103(a) as being unpatentable over Silver et al. in view of York et al. Applicants respectfully disagree.

Silver et al. purport to teach a method for allowing individual developers to modify input method editor (IME) source code on an as-needed basis by integrating a JAVA virtual machine with one or more IMEs. To accommodate user-definable IMEs and components, a virtual machine implements a predefined set of interfaces that allow the IMEs and components to communicate with each other and with the other elements of the virtual machine, namely a windows manager and an input method manager (IMM). The windows manager receives

keyboard commands from the operating system running on the computer system and transfers those commands to the IMM. An IME receives the keyboard commands from the IMM and translates the keyboard commands into a composed character, such as a calligraphic ideogram or an element of a calligraphic ideogram (e.g., a Japanese, Chinese or Korean character). The component receives the composed character from the IME and renders the composed character for display on the display device.

To render the composed characters, however, Silver et al. teach that the windows manager should translate keyboard commands into a standard Unicode format and routes the Unicode keyboard commands to the component or to the IMM, as appropriate (see col. 5, lines 51 – 54). Silver et al. further teach that the IME should include a look-up table of composed characters that are indexed by the various Unicode keyboard commands in order to translate the Unicode keyboard commands into the corresponding composed characters (see col. 6, lines 3 – 21).

However, Silver et al. do not teach the steps of ***passing the character sequence to an IME device driver, the IME device driver converting the character sequence into a first encoding format***; ***passing the character sequence encoded in the first format to an operating system-based IME service module***; and ***from the operating system-based IME service module, calling said platform independent IME to convert said character sequence to a corresponding code point in a second encoding format, the code point being a value assigned to the desired character in a Specification of the second encoding format*** as claimed.

York et al. purport to teach a system for adding functionality to a graphical user interface of a non-Java based, or native, application, using Java. To add the functionality to the user interface of the native application, York et al. teach that new dialog support should be written in the Java programming language. Under the standard protocol, a call to the Java Native Interface (JNI) should block and wait for the function call to complete, then accept a return value. Thus

in the system, the JNI calls do block, as normal, but, once a Java thread is created, control immediately returns to a separate thread where the native application can run. Following the creation of the Java thread, message passing between the thread and the native application is implemented, to "register" the Java dialog with the native application.

After registration, the Java-based dialog performs its intended function as a thread separate from the native code. When the native code receives the information for registering the Java dialog, it determines the handle of the Java dialog and then creates its own "invisible" dialog. The invisible dialog is subsequently used for all message passing to the Java dialog.

The result then is a very seamless integration on a graphical user interface between the Java-based code and the native code. From the perspective of the user, the interface operates and visually appears as if controlled from a single source of code.

However, just as in the case of Silver et al, York et al, do not teach the steps of ***passing the character sequence to an IME device driver, the IME device driver converting the character sequence into a first encoding format***; ***passing the character sequence encoded in the first format to an operating system-based IME service module***; and ***from the operating system-based IME service module, calling said platform independent IME to convert said character sequence to a corresponding code point in a second encoding format, the code point being a value assigned to the desired character in a Specification of the second encoding format*** as claimed.

Hence, Applicants submit that Claim 1, as well as its dependent claims, are allowable over the combined teachings of the applied references. Independent Claims 8, 16 and 20, which incorporate the emboldened-italicized limitations of the above-reproduced Claim 1 are also allowable over the combined teachings of the applied references. Consequently, Applicants once

more respectfully request reconsideration, allowance and passage to issue of the claims in the application.

Respectfully Submitted

By: _____
Volel Emile
Attorney for Applicants
Registration No. 39,969
(512) 306-7969